

# Connect.GPIO v2

MacGruber's Tutorial Series

This guide is intended to give you a quick intro into using my Connect plugin and the ConnectGPIO external app that is intended to run on a Raspberry PI.

## Overview

Overview .....	1
Introduction .....	1
Hardware needed .....	1
Running on Raspberry PI.....	2
Compiling on Raspberry PI.....	2
Integrate the VaM Plugin into your scene .....	3
Circuit Example .....	5

## Introduction

Connect.GPIO is intended to control the GPIO port of a Raspberry PI. The intention is that your VaM scene can take control of whatever devices and things you connect to the GPIO pins. Could be vibrators, suction devices, etc.

It consists of a VaM plugin and a matching server application that runs on the Raspberry PI. The VaM plugin sends commands via network to the application using VaM's trigger system. This should allow you to integrate this into existing scenes easily. Both should find each other automatically once your pressed "Connect" in the VaM plugin UI.

## Hardware needed

You need a Raspberry PI that has network capability. For example, I'm using a Raspberry PI 3B+ which has both wired as well as wireless network capability. Other models might be fine, too. Some of the newer 4B variants might actually be more powerful and cheaper. One of the way cheaper Raspberry Pi Zero variants with Wifi (e.g. W, WH and 2W) might also be already enough, depending on what you want to do with it.

In addition to the PI, you may want additional things:

- Like USB power supply or battery.
- Some kind of case, as handling a raw circuit board can be annoying
- While not officially needed, I would recommend at least a passive cooler.
- During setup you need screen, keyboard and possibly mouse, but once your PI is installed, you can use for example *RemoteDesktop* to connect to it from your Windows PC.
- Possibly one of those starter kits containing things like an experimentation board, cables and a bunch of electric components like LEDs and resistors. Testing with LEDs instead of actual devices can help a lot when building a scene.

## Running on Raspberry PI

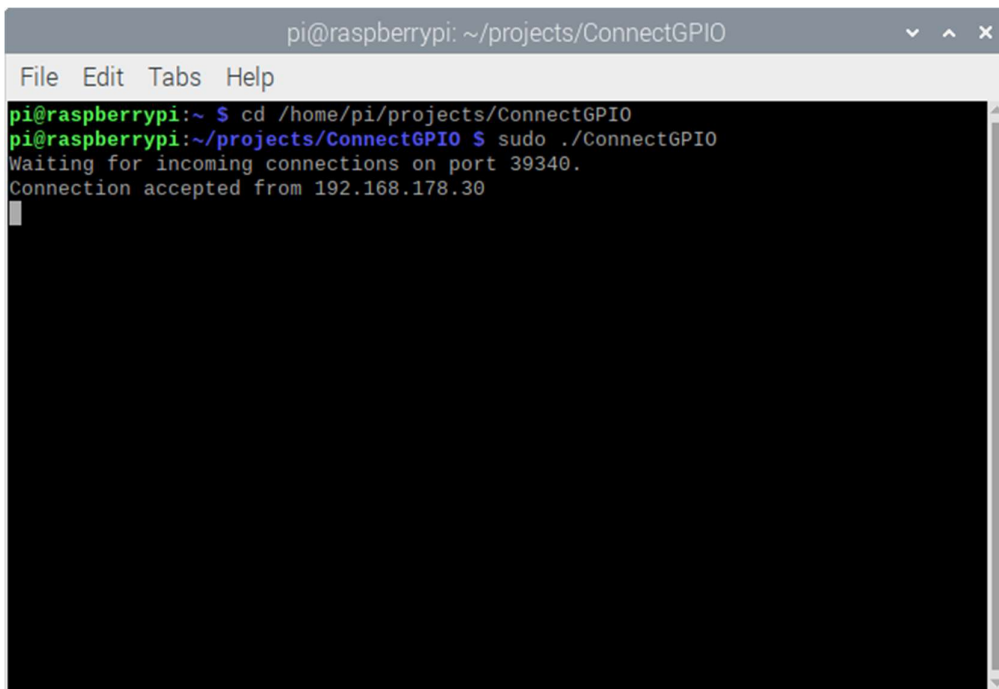
This should be pretty straightforward for any Linux user:

1. Download and extract the package somewhere.
2. ConnectGPIO requires libraries *pthread* and *pigpio* in order to work. However, these should be pre-installed on most Linux OS variants intended for use on a Raspberry PI.
3. Open a terminal window, navigate to the folder, e.g.:

```
cd /home/pi/projects/ConnectGPIO
```

4. Run:

```
sudo ./ConnectGPIO
```



The screenshot shows a terminal window titled "pi@raspberrypi: ~/projects/ConnectGPIO". The terminal output is as follows:

```
pi@raspberrypi:~ $ cd /home/pi/projects/ConnectGPIO
pi@raspberrypi:~/projects/ConnectGPIO $ sudo ./ConnectGPIO
Waiting for incoming connections on port 39340.
Connection accepted from 192.168.178.30
```

## Compiling on Raspberry PI

If you want to make changes to the code, like changing the GPIO pins or the network port being used or some other reason, you can do it like the following. Again, this should be pretty straightforward for any Linux user.

1. To compile you need *g++* and *make*. However, these should be pre-installed on almost any Linux OS variant.
2. Open a terminal window, navigate to the folder, e.g.:

```
cd /home/pi/projects/ConnectGPIO
```

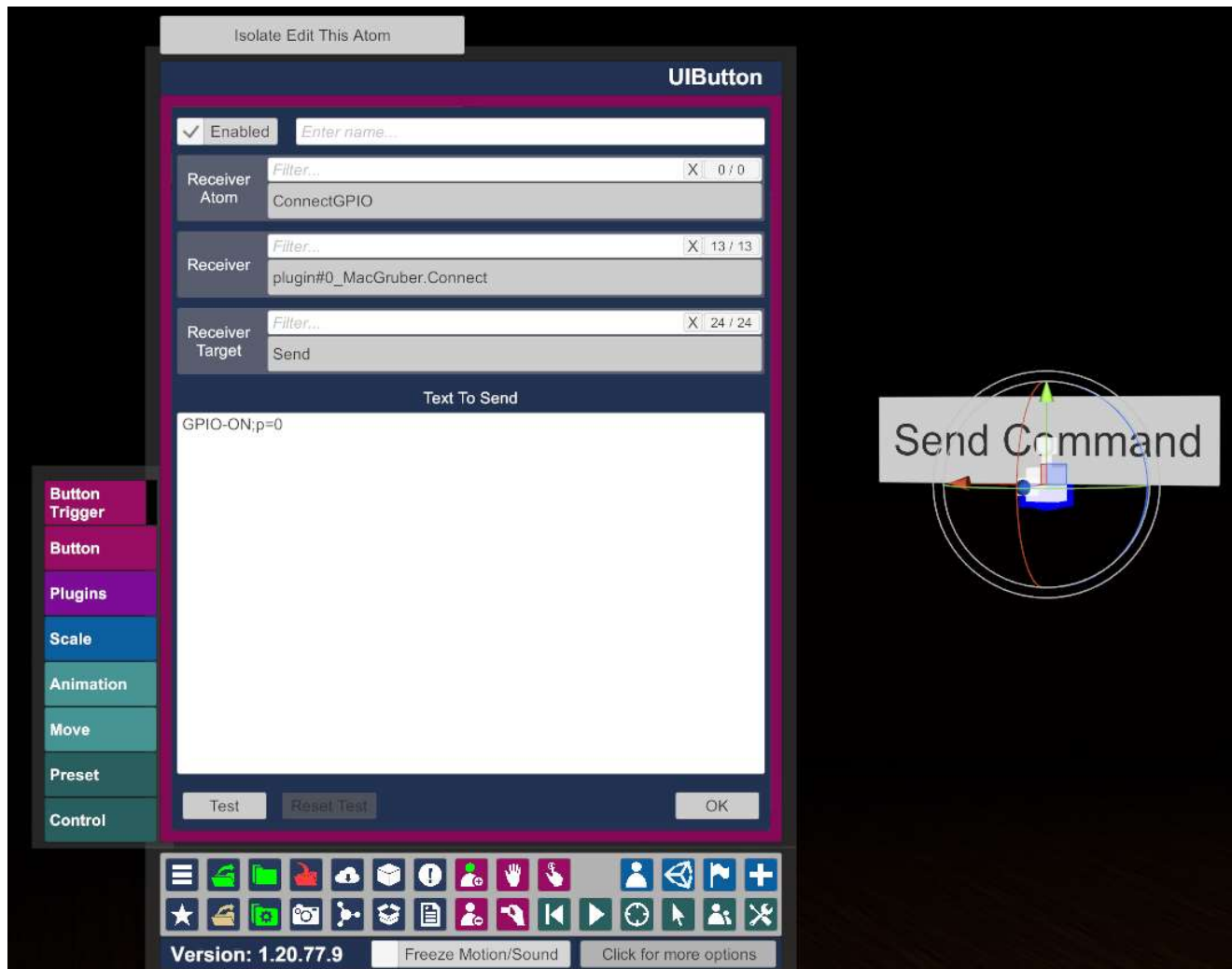
3. Execute the included "makefile" by running:

```
Make
```

Of course, you can also use some Programming IDE like *Geany* or *CodeLite* for development. But it's not required. *Geany* is actually terrible, but pre-installed and good enough for just a few files. However, I had trouble installing *CodeLite*.

## Integrate the VaM Plugin into your scene

On the VaM side you can add the *Connect* plugin to any atom or just add an “Empty” atom and rename it to something that can be easily found. Then you can hook up other things to send commands by using VaM’s trigger system. For example, here we are using an “UIButton” atom, but this works with anything else that uses VaM triggers, including Acidbubbles Timeline, MacGruber LogicBricks and other plugins.



The text you send to the plugin is the name of the command you would like to run on the server, followed by parameters separated by semicolon. Currently supported commands and parameters are:

#### GPIO-ON

Sets specified GPIO pins to “on”, leaves other pins unchanged.

- **p=VALUE** where VALUE is the pin number, between 0 and 9. You can only set a single pin with this.
- **m=VALUE** where VALUE is a bitmask indicating the pins you want to change. For example, *m=1001000000* would set pins 0 and 3. If the bitmask is shorter, the missing pins are assumed zero, so *m=1001* would be equivalent.

#### GPIO-OFF

Same as GPIO-ON, except that specified pins are set to “off”.

#### GPIO

Sets specified GPIO pins to specified value, leaves other pins unchanged.

- **p=VALUE** where VALUE is the pin number, between 0 and 9. You can only set a single pin with this.
- **m=VALUE** where VALUE is a bitmask indicating the pins you want to change. For example, *m=1001000000* would set pins 0 and 3. If the bitmask is shorter, the missing pins are assumed zero, so *m=1001* would be equivalent.
- **v=VALUE** where VALUE indicates to what value pins should be set. If you used p=VALUE, this is just a 0 or 1 value. However, if you used m=VALUE, this is a bitmask as well. For example, using *m=1001000000;v=0001000000* would set pin 0 to “off” and pin 3 to “on”. If the bitmask is shorter, the missing pins are assumed zero, so *m=1001;v=0001* would be equivalent.

Some example commands should make this more clear:

- **GPIO-ON;p=0** Sets pin 0 to “on”.
- **GPIO-ON;m=1001000000** Sets pins 0 and 3 to “on”.
- **GPIO-ON;m=1001** Sets pins 0 and 3 to “on”.
  
- **GPIO-OFF;p=7** Sets pin 7 to “off”.
- **GPIO-OFF;m=1111111111** Sets all 10 pins to “off”.
- **GPIO-OFF;m=0110** Sets pins 1 and 2 to “off”.
  
- **GPIO;p=2;v=1** Sets pin 2 to “on”.
- **GPIO;p=2;v=0** Sets pin 2 to “off”.
- **GPIO;m=1001000000;v=0001000000** Sets pin 0 to “off” and pin 3 to “on”.
- **GPIO;m=1001;v=0001** Sets pin 0 to “off” and pin 3 to “on”.

*Note: The command interpreter is designed to be FAST so you can fire multiple commands per second, if needed. However, that means it is STUPID and can't handle any unexpected symbols like whitespaces in the command. The log in the application will tell you if the command was not understood.*

## Circuit Example

Controlling the two magnet valves of a Suck-o-Mat suction machine. Two Photocouplers (KB817) make sure circuits of both the Raspberry PI as well as the Suck-o-Mat are properly isolated, as both have their own power supply. As common Photocouplers can only handle low amounts of power, two MOSFETs (BS170 TO-92 350mA) are used to amplify the signal. Of course, this could also have been build using mechanical relays.

